FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Thesis of MSc in Mathematics

Unsupervised bilingual dictionary induction with stable GANs

Author: Xuwen Zhang <dlv618@alumni.ku.dk>

Supervisor: Anders Søgaard <soegaard@di.ku.dk>

Submitted on:

Abstract

Contents

Chapter 1

Introduction

1.1 Background

Word embedding is a representation for vocabulary in a document, more specifically, it represents pieces of vocabulary as vectors. Word embedding is able to capture information about how different words are related to each other based on the context in a document. The most popular and powerful technique to learn word embeddings is called Word2Vec which is a shallow neural network, and it was introduced by Mikolov et al. (2013).

Cross-lingual word embeddings means mapping a source monolingual word embedding to another target embedding space in order to align both languages together in the same shared vector space. By Ruder et al. (2017), Cross-lingual word embeddings attract NLP researchers for two reasons:

- It enables computation of cross-lingual word similarities.
- Knowledge can be transferred between different languages, even transfer resource-rich language to resource-insufficient language like English to Finnish.

Broadly speaking, there are two methods to learn cross-lingual embeddings, supervised method and unsupervised method. Conneau et al.(2017) reviewed the supervised method in the beginning of their paper, then contributed a new unsupervised way to learn cross-lingual embeddings. The supervised method assumed that for a given pair of language embeddings, a small dictionary is given such that it would be used as an anchor point to align the two language embeddings. More specifically, assuming the embedding dimension of both embeddings is d, X and Y are matrices with size $d \times n$, and the known dictionary contains n pairs of words $\{x_i, y_i\}_{i \in 1,2,\cdots,n}$ for $x_i \in X$ and $y_i \in Y$, it can be used to learn an optimal linear transformation W between X and Y such that

$$W^* = \underset{W}{\text{argmin}} \, \|WX - Y\|_F \tag{1.1}$$

where W is a $d \times d$ matrix, and $\| \cdot \|_F$ denotes the Frobenius norm. After the optimal transformation is obtained, for any source word s, the translation word

t in target embeddings can be found by searching for the nearest neighbor

$$t = \underset{t}{\text{argmax}} \cos(Wx_s, y_t).$$

The result of optimal W can be improved by orthogonalizing it. The orthogonal improvement is shown by Xing et al. (2015). Thus, the equation (1.1) can be reformed to the Orthogonal Procrustes problem, and a solution would be reached by solving the singular value decomposition(SVD) of YX^T.

$$W^* = \underset{W}{\text{argmin}} \|WX - Y\|_F = UV^T,$$

Where $U\Sigma V^T = \text{SVD}(YX^T)$, the column vectors of U and V are orthonormal, and Σ is a diagonal matrix whose all diagonal elements are positive.

The unsupervised method which Conneau et al. proposed is like this:

- Assume two vector spaces are approximately isomorphic, i.e their exists an invertible linear mapping between them.
- Learn a rough approximation of mapping W by adversarial model. In particular, by generative adversarial networks (GANs).
- Apply the learned mapping W to find word pairs for the most frequent words, and then refine W by Procrusters. The refined W would be used to translate all words in the embedding space.
- In the end, apply W to translate words by searching for the nearest neighbor score, and the nearest neighbor algorithm they used is called cross-domain similarity local scaling(CSLS), which invented by Conneau et al.

1.2 Vanilla GANs and it's improved versions

Generative Adversarial Networks (GANs) are deep neural net architectures that consist of two networks, and GAN's developement is considered a revolutionary advancement in deep learning. The first GAN model proposed by Goodfellow (Goodfellow et al., 2014)[3], it is called vanilla GAN. The idea of GANs was inspired from game theory, in which two models are competing against each other, one is called generator, the other is discriminator, both models will become more and more robust during the training process. More specifically,

- The discriminator is a binary classifier denoted as D which learns to recognize whether the input is coming from a real dataset or a generator's output. Formally, D calculates the probability of the sample coming from the generator G or the real input data for each iteration. Usually the label defined as 1 for real data and 0 for synthetic data.
- The generator denoted as G, takes random noise from the latent space, and outputs synthetic samples. The training purpose for the generator

is to approximate the distribution of real data as close as possible, at the same time the probability for fooling the discriminator would increase too, and this makes higher the probability that the discriminator would treat the synthetic output from the generator as real

To measure the similarity between the distributions of generated samples and real samples, Jensen-Shannon (JS) divergence is used as the measurement instead of Kullback–Leibler (KL) divergence, because JS divergence is symmetric and smoother than KL divergence.

GANs are able to generate data from scratch based on the given sample. The main application area of GANs is computer vision, e.g., creating animated characters (Jin et al. 2017) which could be applied by game developing or animation production companies, high resolution images production based on given low resolution images (Ledig et al. 2016), and new video sequence generation (Vondrick et al. 2016) .

GANs can also be applied to other domains such as music generation (Fedus et al. 2018), text generation (Mogren. 2016), or even text to image synthesis (Reed at el. 2016) etc. These applications show that GANs can be adapted immediately for commercial purpose.

Although vanilla GANs achieved great success, there are several disadvantages which makes vanilla GANs' training procedure unstable,

- Mode collapse happens frequently, which means the generator only learns limited distributions of real data.
- Lack of metric that can show the training process, therefore the training is fully manual.
- Non- convergence, i.e. it is difficult for the model to reach the Nash equilibrium due to oscillating change of gradients.
- Gradient vanished, which means the generator's weights stop updating due to the discriminator performing too well.
- Highly sensitive for tuning hyperparameters.

Several improved versions of GANs attempt to solve the above issues, e.g, Wasserstain GAN (Arjovsky et al. 2017), in which the Earth Mover (Wasserstein) Distance is applied to substitute minimize the JS divergence as the new loss function to make gradient more stable due to the EM distance is much smoother. Although the EM distance has nice properties, it is almost impossible to calculate, therefore, the Kantorovich-Rubinstein duality (Villani. 2008) algorithm is applied, where all weights in the discriminator are constrained to a 1-Lipschitz function, i.e., all weights in the discriminator must be clipped in a bounded interval. However, weight clipping will bring new issues, even Arjovsky pointed out in the original Wasserstein GAN paper that it is a horrible method to enforce a Lipschitz constraint. Instead of weight clipping, the gradient penalty (Gulrajani, et al. 2017) enforces Lipschitz by penalizing the norm of the discriminator's gradient. Our best models are obtained from this algorithm. Another algorithm is CT-GAN (Wei, et al. 2018), where a consistency term is added to the WGAN with gradient penalty.

We focused on the unsupervised part of the MUSE in this thesis and do the following:

1. Try to reproduce the results of Conneau et al for several language pairs, and use the reproduced results as benchmark.

2. Implement the improved GANs which are mentioned previously, and compare our result with the benchmark. However, since we found that the WGAN-GP applied MUSE has the best performance, we focused on it mostly.

3. Analysing the results.

Chapter 2

Vanilla GAN

This chapter shows how Goodfellow et all interpret the mechanism of vanilla GAN, and some of the issues Arjovsky et al explored.

2.1 Working mechanism

The training purpose for the generator G is to make the distribution of it's output more and more close to real data, the goal of the discriminator D is to achieve a Nash equilibrium for which it is impossible to distinguish whether it's input comes from real or generated dataset. Formally speaking, the task of the discriminator D is to maximize

$$\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})]$$

which means increasing the probability to identify samples coming from real data, as well as to reduce the probability $D(G(\boldsymbol{x}))$ to zero by maximizing

$$\mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[\log(1 - D(G(\boldsymbol{x})))].$$

Thus, the objective function of the discriminator is

$$\max_D L(D) = \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_g(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{x})))].$$

At the same time, the task for training the generator G is to increase the probability of the output of D to assign labels to fake data, therefore the objective function of the generator is

$$\min_G L(G) = \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[\log(1 - D(G(\boldsymbol{x})))].$$

Since D and G are playing a minimax game, the final objective function is

$$\min_G \max_D L(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_g(\boldsymbol{z})}[(1 - \log D(G(\boldsymbol{x})))]. \quad (2.1)$$

The notation $p_r(\boldsymbol{x})$ means the probability distribution for real data, $p_z(\boldsymbol{x})$ denotes generated fake data.

Proposition 2.1: [Proposition 1 (Goodfellow et all., 2014)]
For fixed G, the optimal value for D is:

$$D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}. \tag{2.2}$$

$Proof$:

Here is a summary of the proof by the original author:

Since the loss function in continuous probability distribution is

$$L(G, D) = \int_{\mathcal{X}} \Big(p_r(\boldsymbol{x})\log(D(\boldsymbol{x})) + p_{g(\boldsymbol{x})}\log(1 - D(\boldsymbol{x}))\Big)dx$$

the maximum value of $L(D, G)$ can be achieved by solving the function inside the integral. For convenience, define the function inside the integral as

$$f(y) = p_r(x)\log y + p_g(x)\log\big(1 - y\big),$$

and take the derivative of loss with respect to x and set it to zero, then get

$$\begin{aligned}\frac{df(y)}{dy} &= p_r(x)\frac{1}{y} - p_g(x)\frac{1}{1-y} \\ &= \frac{p_r(x) - y\big(p_r(x) + p_g(x)\big)}{y(1-y)} \\ &= 0 \\ \Longrightarrow y &= \frac{p_r(x)}{p_r(x) + p_g(x)}\end{aligned}$$

Apply the above to a vector variable x.
Thus, the $D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})+p_g(\boldsymbol{x})} \in [0, 1]$. □

By proposition 1 above, The equation (2.1) can be rewritten as

$$\begin{aligned}L(G, D^*) = &\max_D L(G, D) \\ =&\mathbb{E}_{\boldsymbol{x}\sim p_r}[\,\log D^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[(1 - \log D^*(\boldsymbol{x}))\,] \\ =&\mathbb{E}_{\boldsymbol{x}\sim p_r}\Big[\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\Big] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\big[\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\big]\end{aligned}$$

Theorem 2.1: [Theorem 1 (Goodfellow et all., 2014)]

The global minimum of $L(G, D^*)$ can be reached if and only if $p_r(\boldsymbol{x}) = p_g(\boldsymbol{x})$, at this point, $C(G) = -2\log 2$ and $D^*(\boldsymbol{x}) = 0.5$.

$Proof$:

Here is a summary of the proof by the original author:

By equation (2.2), if $p_r(\boldsymbol{x}) = p_g(\boldsymbol{x})$, then $D^*(\boldsymbol{x}) = 0.5$, substituting this value to equation (4) we have

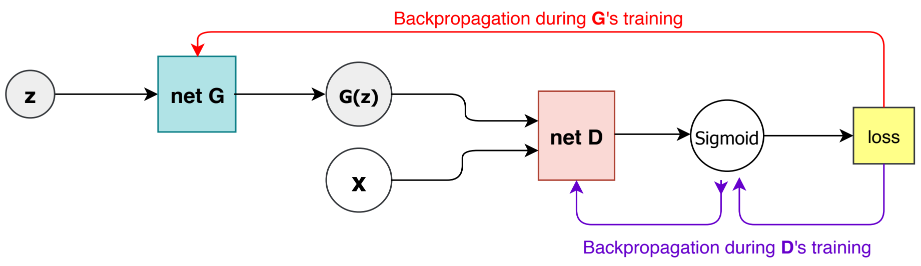
$$\begin{aligned}\min_G L(G, D^*) &= \int_{\mathcal{X}} \Big(p_r(\boldsymbol{x})\log\frac{1}{2} + p_{g(\boldsymbol{x})}\log\frac{1}{2}\Big)dx \\ &=\log\frac{1}{2}\Big(\int_{\mathcal{X}} p_r(\boldsymbol{x})dx + \int_{\mathcal{X}} p_{g(\boldsymbol{x})}dx\Big) \\ &=2\log\frac{1}{2} \\ &= -\ 2\log2.\end{aligned}$$

to show $-2\log2$ is the optimal value for $L(G, D^*)$, we need to prove they are equal to each other. Subtracting one from the other and by applying proposition 1 we get

$$\begin{aligned}L(G, D^*) - \big(-2\log2\big) =&2\log2 + \int_{\mathcal{X}} \Big(p_r(\boldsymbol{x})\,\log\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \\ &+p_g(\boldsymbol{x})\,\log\frac{p_g(\boldsymbol{x})}{p_r(x) + p_g(\boldsymbol{x})}\Big)dx \\ =&\Big(\log2 + \int_{\mathcal{X}} p_r(\boldsymbol{x})\,\log\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big) \\ &+\Big(\log2 + \int_{\mathcal{X}} p_g(\boldsymbol{x})\,\log\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big) \\ =&\Big(\int_{\mathcal{X}} p_r(\boldsymbol{x})\log2\,dx + \int_{\mathcal{X}} p_r(\boldsymbol{x})\,\log\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big) \\ &+\Big(\int_{\mathcal{X}} p_g(\boldsymbol{x})\log2\,dx + \int_{\mathcal{X}} p_g(\boldsymbol{x})\,\log\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big) \\ =&\int_{\mathcal{X}} p_r(\boldsymbol{x})\log\frac{2 \cdot p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx + \int_{\mathcal{X}} p_g(\boldsymbol{x})\log\frac{2 \cdot p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx \\ =&KL\Big(p_r||\frac{p_r + p_g}{2}\Big) + KL\Big(p_g||\frac{p_r + p_g}{2}\Big) \\ =&2JS(p_r||p_g)\end{aligned}$$

Since the Jensen-Shannon divergence equals to zero when the two probability distributions p_r and p_g are equal everywhere, then, the global optimal value for the loss is - 2 log2 and $D^*(\boldsymbol{x}) = 0.5$. □

Below is the architecture draft of vanilla GAN and the training algorithm in the original paper (Goodfellow et all., 2014)[3].



Backpropagation during G's training
z
net G
G(z)
x
net D
Sigmoid
loss
Backpropagation during D's training

Figure 1: The structure of Vanilla GAN

Algorithm 2.1: Minbatch training algorithm uses stochastic gradient descent as optimizer, n is a hyperparameter which for each generator's iteration, trains discriminator D for n iterations. The original paper used $n = 1$.

```
1: for i = 1, 2, . . . , number of training iterations do
2:     for training discriminator $n$ iterations do
3:         Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
4:         Sample $k$ minibatch input examples from data distribution $p_r(\boldsymbol{x})$
5:         Backpropagating all weights in $D$ by maximizing the loss:
               $$\nabla_{W_D} \frac{1}{k} \sum_{i=1}^{k} \left[\log D(\boldsymbol{x}^{(i)}) + \log\left(1 - D\left(G(\boldsymbol{z}^{(i)})\right)\right)\right].$$
6:     end for
7:     Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
8:     Backpropagating all weights in $G$ by minimizing the loss:
           $$\nabla_{W_G} \frac{1}{k} \sum_{i=1}^{k} \log\left(1 - D\left(G(\boldsymbol{z}^{(i)})\right)\right).$$
9: end for
```

However, Goodfellow et all pointed out that in practice, when the training has just started, the distributions of $\mathbb{P}_r$ and $\mathbb{P}_g$ may have a huge difference, in this case, D can easily distinguish whether the data comes from training data or synthetic. Therefore, $\log\big(1 - D(G(\boldsymbol{z}))\big)$ saturates. Maximizing $\log D(G(\boldsymbol{z}))$ can provide more robust gradients in the early training instead of minimizing $\log\big(1 - D(G(\boldsymbol{z}))\big)$.

2.2 Issues

By theorem 1, the minimum of the discriminator's loss is

$$L(G, D^*) = 2JS(\mathbb{P}_r || \mathbb{P}_g) - 2\log 2$$

However, the discriminator's loss will converge to zero in practice . Numerically, 2log2 $= 1.386294$, and this means that $JS(\mathbb{P}_r||\mathbb{P}_g)$ will converge to 0.693 instead of zero, and at this point the distributions of $\mathbb{P}_r$ and $\mathbb{P}_g$ are very similar.

By Arjovsky et al [20], the reason which causes the situation above to happen is that

- Either the distributions are discontinuous, in other words, the probability density function does not exist.
- Or both distributions have disjoint supports.

There is a possible reason which makes the distributions discontinuous, that is both of their supports to lie on lower dimensional manifolds. Arjovsky et al [20] clarified that the term continuous means absolutely continuous. More specifically, for an absolutely random variable, it has the property that if $P(X \in B) = 0$ then B has 0 Lebesgue measure. Whereas a random variable is continuous if $P(X = x) = 0 \;\; \forall x \in X$, where x is a point. Absolutely continuous implies continuous since the Lebesgue measure of the set of points is 0, but not vise versa. Moreover, If a random variable has support on a low dimensional manifold, it is not absolutely continuous.

Narayanan et al. (2010) claimed that the support of the distribution of $\mathbb{P}_r$ lies on a low dimensional manifold. For $\mathbb{P}_g$, if dim($\mathcal{Z}$) $<$ dim($\mathcal{X}$), then $\mathbb{P}_g$ is not continuous. This concept is mathematically formalized in the following lemma.

Lemma 1: [Lemma 1. (Arjovsky et al. 2017)] Assume $g : \mathcal{Z} \mapsto \mathcal{X}$ be a neural network, which is a function composed by affine transformations with non-linear activation functions, then a countable union of manifolds which contains $g(\mathcal{Z})$ would have dimension less than or equal to dim($\mathcal{Z}$). Thus, the measure of $g(\mathcal{Z})$ would be zero in $\mathcal{X}$ if dim($\mathcal{Z}$) $<$ dim($\mathcal{X}$).

Since the generator G is a neural network, this lemma states that if the input dimension of $\boldsymbol{z}$ is less than $g(\boldsymbol{z})$, then the learning ability of $g(\boldsymbol{z})$ will be limited, only a small part of the input of the distribution can be learned by $g(\boldsymbol{z})$ due to the zero measure of $g(\boldsymbol{z})$.

Lemma 2: (Smooth Urysohn's Lemma) [Urysohn's lemma]
Let $\mathcal{M}$ and $\mathcal{S}$ be two compact and disjoint subsets in a normal topological space $(\mathcal{X}, \mathcal{T})$, then there exists a function $f : \mathcal{X} \mapsto [0, 1]$ which is smooth such that $f(x) = 0 \; \forall x \in \mathcal{S}$ and $f(x) = 1 \; \forall x \in \mathcal{M}$.

The next theorem states that if the supports of two distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ are disjoint, there exists a perfect discriminator, along with vanishing gradients. In this case, when D trained too well, G's weights stop updating and make G to stop approximating.

Theorem 3: [Theorem 2.1 (Arjovsky et al. 2017)]

Assume the supports of $\mathbb{P}_r$ and $\mathbb{P}_g$ are included on disjoint compact subsets $\mathcal{M}$ and $\mathcal{S}$ respectively, then there exists a completely accurate smooth discriminator $D^* : \mathcal{X} \mapsto [0,1]$ which has 0 error, plus $\nabla_{\boldsymbol{x}} D^*(\boldsymbol{x}) = 0,\ \forall \boldsymbol{x} \in \mathcal{M} \cup \mathcal{S}$.

$Proof$:

By assumption, $\mathcal{M}$ and $\mathcal{S}$ are disjoint and compact. Supposed $d(\mathcal{M}, \mathcal{S}) = \epsilon > 0$ is the distance between $\mathcal{M}$ and $\mathcal{S}$, then define another two sets

$$\mathcal{M}' = \{\boldsymbol{x} : d(\boldsymbol{x}, \mathcal{M}) \le \frac{\epsilon}{3}\}$$
$$\mathcal{S}' = \{\boldsymbol{x} : d(\boldsymbol{x}, \mathcal{S}) \le \frac{\epsilon}{3}\}$$

therefore $\mathcal{M}'$ and $\mathcal{S}'$ are also both disjoint and compact. Thus, by applying Smooth Urysohn's Lemma, exists a continuous function $D^* : \mathcal{X} \mapsto [0,1]$ such that $D^*\big|_{\mathcal{S}'} \equiv 0$ and $D^*\big|_{\mathcal{M}'} \equiv 1$. Since $\forall \boldsymbol{x} \in \text{supp}(D^*\big|_{\mathbb{P}_r}) = \{\boldsymbol{x} \in \mathbb{P}_r \big| D^* \neq 0\}$, $\log D^*(\boldsymbol{x}) = 0$, and $\log\big(1 - D^*(\boldsymbol{x})\big) = 0\ \forall \boldsymbol{x} \in \text{supp}(D^*\big|_{\mathbb{P}_g})$, therefore the discriminator has zero error and has become optimal. In addition, for proving $D^*(\boldsymbol{x}) = 0$, first define $\boldsymbol{x} \in \mathcal{M} \cup \mathcal{S}$, then for $\boldsymbol{x} \in \mathcal{M}$, there exists an open ball $\mathcal{B}(\boldsymbol{x}, \frac{\epsilon}{3}) = \mathcal{B}'$ such that $D^*\big|_{\mathcal{B}'}$ is constant. Thus $\nabla_{\boldsymbol{x}} D^* \equiv 0$. For $\boldsymbol{x} \in \mathcal{S}$, following the same procedure will finish the proof. □

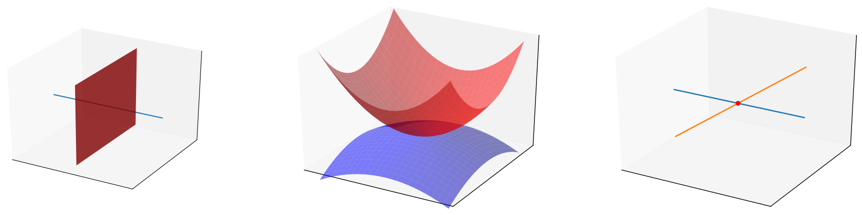
This theorem says, if the intersection of the supports of $\mathbb{P}_r$ and $\mathbb{P}_g$ is empty, then there exists a perfect discriminator which is able to separate $\mathbb{P}_r$ and $\mathbb{P}_g$, and D's gradient will be vanished on the union of the two supports. This makes G stop learning when D trained too well.

Definition 1: (Transversal intersection) [Definition 2.2 (Arjovsky et al. 2017)] Assume $\mathcal{M}$ and $\mathcal{S}$ are two submanifolds of $\mathcal{F}$, and $\mathcal{F} = \mathbb{R}^n$. If $\forall p \in \mathcal{M} \cap \mathcal{S}$

$$T_p\mathcal{M} + T_p\mathcal{S} = T_pF$$

then we say that $\mathcal{M}$ and $\mathcal{S}$ intersect transversally.

$T_p\mathcal{M}$ means the tangent space of $\mathcal{M}$ at point p. Moreover, if two submanifolds do not have intersection, then they are transversal.



(a) transverse (b) non-transverse (c) non-transverse

Figure 1: examples of transversal intersection

Definition 2: (Perfect align) [Definition 2.2 (Arjovsky et al. 2017)] Assume $\mathcal{M}$ and $\mathcal{S}$ are two manifolds without boundary. We say $\mathcal{M}$ and $\mathcal{S}$ perfectly align if there exists $\boldsymbol{x} \in \mathcal{M} \cap \mathcal{S}$ such that $\mathcal{M}$ and $\mathcal{S}$ are non-transversal at $\boldsymbol{x}$.

Lemma 2: [Lemma 2. (Arjovsky et al. 2017)]
Assume $\mathcal{M}$ and $\mathcal{S}$ are two regular submanifolds of $\mathbb{R}^n$, and they do not have a full dimension. Suppose ϵ and ϵ' are two independent continuous random variables, defining two perturbed manifolds $\hat{\mathcal{M}} = \mathcal{M} + \epsilon$ and $\hat{\mathcal{S}} = \mathcal{S} + \epsilon'$. Then

$$\mathbb{P}_{\epsilon,\epsilon'}\left(\hat{\mathcal{M}} \text{ and } \hat{\mathcal{S}} \text{ not perfectly align }\right) = 1.$$

This lemma means that any small perturbations can make two low dimension manifolds become non-perfectly aligned, i.e., transversally intersect.

Lemma 3: [Lemma 3. (Arjovsky et al. 2017)]
Assume $\mathcal{M}$ and $\mathcal{S}$ are two non-full dimension and non-perfectly aligned regular submanifolds of $\mathbb{R}^n$. Let $\mathcal{M} \cap \mathcal{S} = \mathcal{P}$.

- If $\mathcal{M}$ and $\mathcal{S}$ have boundary, then $\mathcal{P}$ is a union of no more than four strictly low dimensional manifolds.
- If $\mathcal{M}$ and $\mathcal{S}$ without boundary. then $\mathcal{P}$ is still a manifold which has strictly lower dimension than $\mathcal{M}$ or $\mathcal{S}$.

$\mathcal{P}$ has measure zero in $\mathcal{M}$ and $\mathcal{S}$ in both cases.

By lemma 2, two manifols can be non-perfectly aligned by any subtle disturbation, therefore by lemma 3, their intersection has zero measure.

Theorem 4: [Theorem 2.2 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are two distributions, their supports contained respectively in two closed lower dimensional manifolds $\mathcal{M}$ and $\mathcal{S}$ which are non-perfectly aligned. In addition, suppose $\mathbb{P}_r$ and $\mathbb{P}_g$ are continuous, which means that for any set $A \in \mathcal{M}$ with zero measure it is $\mathbb{P}_r(A) = 0$, similar for $\mathbb{P}_g$. Then there is an full accuracy optimal discriminator $D^* : \mathcal{X} \mapsto [0,1]$ which can distinguish $\mathbb{P}_r$ and $\mathbb{P}_g$. Moreover, $\forall \boldsymbol{x} \in \mathcal{M}$ or $\forall \boldsymbol{x} \in \mathcal{S}$, $\nabla_{\boldsymbol{x}} D^*(\boldsymbol{x}) = 0$ and D^* is smooth in a neighborhood.

In theorem 3, we assumed that the supports of both $\mathcal{M}$ and $\mathcal{S}$ are disjoint. However theorem 4 says, even in the condition that the supports of $\mathcal{M}$ and $\mathcal{S}$ are joint and non-perfectly aligned,there still exists a perfect discriminator which can separate $\mathcal{M}$ and $\mathcal{S}$, which also makes the generator's gradient vanish.

Theorem 5: [Theorem 2.3 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are two distributions, their supports contained respectively in two lower dimensional manifolds $\mathcal{M}$ and $\mathcal{S}$ which are non-perfectly aligned. Furthermore, assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are continuous in their manifolds. Then

- $JSD(\mathbb{P}_r \| \mathbb{P}_g) = \ln 2$
- $KL(\mathbb{P}_r \| \mathbb{P}_g) = +\infty$
- $KL(\mathbb{P}_g \| \mathbb{P}_r) = +\infty$

This theorem indicates that under the same condition of theorem 4, a perfect discriminator can be trained, then the Jensen Shannon divergence between two distributions will get close to ln 2. In addition, KL divergence is not available as a metric to evaluate GAN's performance, thus finding a useful metric is the new goal. There are some examples about KL divergence in the next chapter.

Theorem 6: [Theorem 2.4 (Arjovsky et al. 2017)]
Assume $g_\theta : Z \mapsto \mathcal{X}$ is a differentiable function which induce the distribution $\mathbb{P}_g$ and let $\mathbb{P}_r$ be the input data distribution. Let D be a differentiable discriminator. If the conditions in theorem 2.1 and 2.2 are satisfied, $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\|J_\theta g_\theta(z)\|_2^2\right] \leq M^2$ and $\|D - D^*\| < \epsilon$, then

$$\left\|\nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D(g_\theta(\boldsymbol{z}))\right)\right]\right\|_2 < M\frac{\epsilon}{1-\epsilon}$$

Proof:

Since the theorem 2.1 and 2.1 satisfied, $\nabla_{\boldsymbol{x}} D^*(\boldsymbol{x}) = 0$. Therefore, by Jensen's inequality and the chain rule

$$\begin{aligned}
\left\|\nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D(g_\theta(\boldsymbol{z}))\right)\right]\right\|_2^2 &\leq \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\|\nabla_\theta D\big(g_\theta(\boldsymbol{z})\big)\|_2^2}{|1 - D\big(g_\theta(\boldsymbol{z})\big)|^2}\right] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\|J_\theta g_\theta(\boldsymbol{z})\nabla_{\boldsymbol{x}} D\big(g_\theta(\boldsymbol{z})\big)\|_2^2}{|1 - D\big(g_\theta(\boldsymbol{z})\big)|^2}\right] \\
&\leq \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\|\nabla_{\boldsymbol{x}} D\big(g_\theta(\boldsymbol{z})\big)\|_2^2 \|J_\theta g_\theta(\boldsymbol{z})\|_2^2}{|1 - D\big(g_\theta(\boldsymbol{z})\big)|^2}\right] \\
&\leq \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\left(\|\nabla_{\boldsymbol{x}} D^*\big(g_\theta(\boldsymbol{z})\big)\|_2 + \epsilon\right)^2 \|J_\theta g_\theta(\boldsymbol{z})\|_2^2}{\left(|1 - D^*\big(g_\theta(\boldsymbol{z})\big)| - \epsilon\right)^2}\right] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\frac{\epsilon^2 \|J_\theta g_\theta(\boldsymbol{z})\|_2^2}{\left(1 - \epsilon\right)^2}\right] \\
&\leq M^2 \frac{\epsilon^2}{(1-\epsilon)^2}
\end{aligned}$$

After taking the square root of each side, we get the final result

$$\left\|\nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D(g_\theta(\boldsymbol{z}))\right)\right]\right\|_2 < M\frac{\epsilon}{1-\epsilon}$$

$\square$

Corollary 1: [Corollary 2.1 (Arjovsky et al. 2017)]
Assume all conditions from theorem 6 are satisfied, then

$$\lim_{\|D-D^*\|\to 0} \nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\big[\log\big(1 - D(g_\theta(\boldsymbol{z}))\big)\big] = 0$$

This means the generator's gradient is bounded, in other words,the more close the discriminator D is to it's optimal, the smaller the gradient of the generator is. Therefore, in this situation, the generator stops learning the input distribution .

In practice, the alternative loss function log$D(G(\boldsymbol{z}))$ does not cause the gradient to vanish, but still it's gradient suffers from unstable update.

Theorem 7: [Theorem 2.5 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are two distributions with density function P_r and P_g respectively. $D^* = \frac{P_r}{P_{g_{\theta_0}}+P_r}$ is the optimal discriminator with a fixed value θ_0. Then

$$\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\big[-\nabla_\theta \log D^*(g_\theta(z))\big|_{\theta=\theta_0}\big]$$

The gradient of the new alternative loss function has an inverse KL-divergence and minus JS-divergence between $\mathbb{P}_r$ and $\mathbb{P}_g$. The inverse KL-term can penalize largely different samples by assigning high costs to them, but low costs for mode collapse. Since JS divergence is symmetric, it can not change the problem.

Chapter 3

Wasserstein GAN

The way vanilla GAN works and why it's hard to train has already shown up in the previous chapter. Arjovsky et al. (2017) theoretically explored the drawbacks of Vanilla GAN and applied the Earth Mover Distance(EMD) to introduce a new GAN called Wasserstein GAN which tries to solve them, e.g., mode collapse, and lack of metric that informs us about the training progress. However, even though Wasserstein GAN has better performance than vanilla GAN in image generation, it still has some flaws such as weight clipping.

3.1 Several probability metrics

There are several formulas to measure the similarity between two probability distributions over the same random variable x. Assume $\mathbb{P}_r(\boldsymbol{x})$ and $\mathbb{P}_g(\boldsymbol{x})$ are two distributions.

Kullback-Leibler (KL) divergence (continuous form):

$$KL(\mathbb{P}_r||\mathbb{P}_g) = \int_{\mathcal{X}} \log\Big(\frac{P_r(\boldsymbol{x})}{P_g(\boldsymbol{x})}\Big) P_r(\boldsymbol{x}) dx,$$

it's lower bounded by 0. KL Divergence reaches minimum zero while $P_r(x) = P_g(x)$ for all $x \in \mathcal{X}$, is asymmetric . The discrete form of KL divergence is :

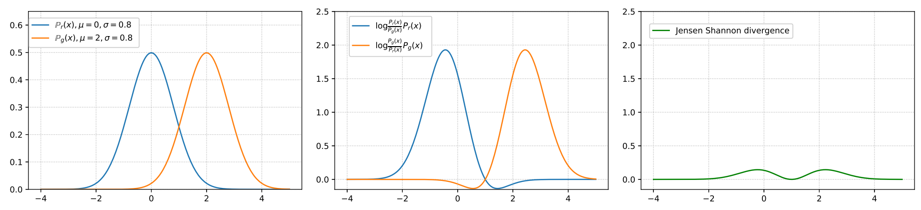
$$\begin{aligned} KL(\mathbb{P}_r||\mathbb{P}_g) =& H(p) - H(p,q) \\ =& \sum_x \log\Big(\frac{P_r(\boldsymbol{x})}{P_g(\boldsymbol{x})}\Big) P_r(x) \\ =& -\sum_x \log\Big(\frac{P_g(\boldsymbol{x})}{P_r(\boldsymbol{x})}\Big) P_r(x), \end{aligned}$$

where $H(p,q)$ is the cross entropy between p and q. KL divergence is asymmetric.

Jensen-Shannon(JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = \frac{1}{2} KL(\mathbb{P}_r||\mathbb{P}_m) + \frac{1}{2} KL(\mathbb{P}_g||\mathbb{P}_m)$$

where $\mathbb{P}_m = \frac{\mathbb{P}_r+\mathbb{P}_g}{2}$, the JS divergence is symmetric, and bounded in $[0, 1]$. It is symmetric, i.e. $JS(\mathbb{P}_r, \mathbb{P}_g) = JS(\mathbb{P}_g, \mathbb{P}_r)$.



(a) $\mathbb{P}_r(\mathcal{X})$ and $\mathbb{P}_g(\mathcal{X})$ (b)$KL(\mathbb{P}_r||\mathbb{P}_g)$ and $KL(\mathbb{P}_g||\mathbb{P}_r)$ (c) Jensen Shannon divergence

Figure 1: Two different normal distributions with corresponding KL and JSD

Earth Mover Distance(EMD):

The last measure is The Earth Mover's Distance (EMD), also called Wasserstein Distance. Informally, assuming there are two different distribution piles of dirt $\mathbb{P}$ and $\mathbb{Q}$ with same total amount the EMD can be interpreted as the minimal cost for moving one distribution pile to another to make both $\mathbb{P}$ and $\mathbb{Q}$ have the same distribution. Thus we can calculate EMD as the sum of work flow for moving piles from $\mathbb{P}$ to $\mathbb{Q}$, for each move, the mass times the distance. Sometimes there are lots of ways for moving one distribution to another. Therefore to find the way with the minimal cost is an optimization problem.

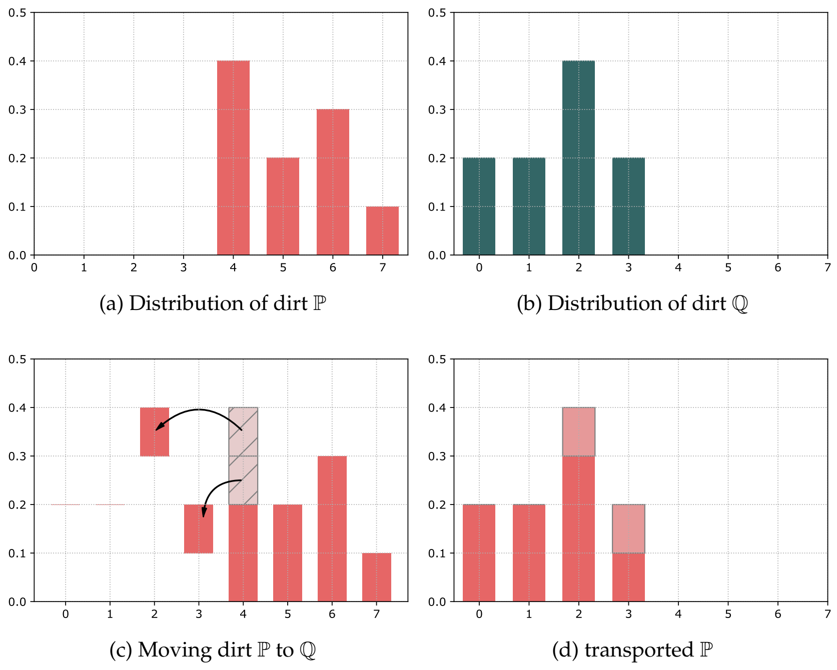
Definition of discrete case:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \sum_{\boldsymbol{x},\boldsymbol{y}} \|\boldsymbol{x}-\boldsymbol{y}\|\gamma(\boldsymbol{x},\boldsymbol{y}) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma}\left[\|\boldsymbol{x}-\boldsymbol{y}\|\right]$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of all combinations of joint distributions between $\mathbb{P}_r$ and $\mathbb{P}_g$. Since, γ means the percentage of earth should be moved from $\boldsymbol{x}$ to $\boldsymbol{y}$, thus $\mathbb{P}_g(\boldsymbol{y}) = \sum_{\boldsymbol{x}} \gamma(\boldsymbol{x}, \boldsymbol{y})$ and $\mathbb{P}_r(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \gamma(\boldsymbol{x}, \boldsymbol{y})$, which means $\mathbb{P}_g(\boldsymbol{y})$ and $\mathbb{P}_r(\boldsymbol{x})$ are marginal distributions of $\gamma(\boldsymbol{x}, \boldsymbol{y})$ respectively.

Here is a simple example of how Earth Mover's Distance works, suppose X and Y are two different discrete random variables over the spaces $\mathbb{P}$ and $\mathbb{Q}$, then the EMD from $\mathbb{P}$ to $\mathbb{Q}$ is

$$\begin{aligned}\text{EMD}(\mathbb{P}, \mathbb{Q}) =& 0.1 * |4-3| + 0.1 * |4-2| + 0.2 * |4-0| \\ &+ 0.2 * |5-1| + 0.3 * |6-2| + 0.1 * |7-3| \\ =& 3.5\end{aligned}$$


(a) Distribution of dirt $\mathbb{P}$

(b) Distribution of dirt $\mathbb{Q}$

(c) Moving dirt $\mathbb{P}$ to $\mathbb{Q}$

(d) transported $\mathbb{P}$

Figure 1: Transporting discrete probability distributions from $\mathbb{P}$ to $\mathbb{Q}$

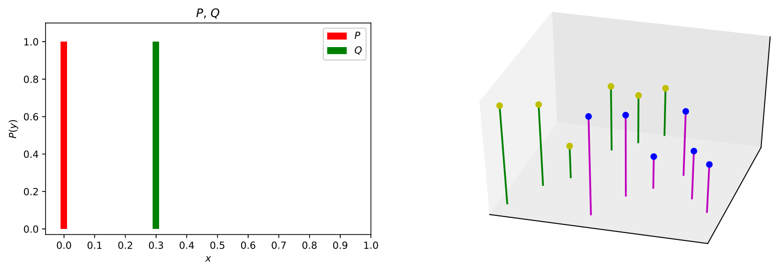
3.2 Why is Wasserstain GAN better

Here is an example of why it is better to use Wasserstein:

Let Y be the random variable $Y \sim U[0,1]$, $\mathbb{P}$ and $\mathbb{Q}$ are two probability distributions, where $\mathbb{P}$ is the distribution on $(0,Y) \in \mathbb{R}^2$, $\mathbb{Q}$ is $(\theta, Y) \in \mathbb{R}^2$ for $\theta \in [0,1]$. If $\theta \neq 0$ then we have:

$$
\begin{aligned}
&KL(\mathbb{P}||\mathbb{Q}) = \sum_{x=0, Y\sim U[0,1]} 1 \times \ln\frac{1}{0} = \infty \\
&KL(\mathbb{Q}||\mathbb{P}) = \sum_{x=\theta, Y\sim U[0,1]} 1 \times \ln\frac{1}{0} = \infty \\
&JS(\mathbb{P}||\mathbb{Q}) = \frac{1}{2}KL\Big(\mathbb{P}||\frac{\mathbb{P}+\mathbb{Q}}{2}\Big) + \frac{1}{2}KL\Big(\mathbb{Q}||\frac{\mathbb{Q}+\mathbb{P}}{2}\Big) \\
=&\frac{1}{2}\Big(\sum_{x=0, Y\sim U[0,1]} 1 \times \ln\frac{1}{0.5} + \sum_{x=\theta, Y\sim U[0,1]} 1 \times \ln\frac{1}{0.5}\Big) \\
=&\ln 2 \\
&W(\mathbb{P},\mathbb{Q}) = |\theta|
\end{aligned}
$$

If $\theta = 0$, then $\mathbb{P}$ and $\mathbb{Q}$ completely overlap. This indicates that in most cases, KL divergence is impossible to calculate when two distributions are discrete.



2d 3d

Figure 1: 2d and 3d plot of non-overlapped probability distributions $\mathbb{P}$ and $\mathbb{Q}$

Lebesgue's Dominated Convergence Theorem.
Let $\{f_n\}$ be a sequence of real-valued measurable functions on a measure space (S, Σ, μ). Suppose that the sequence converges pointwise to a function f

$$\left|f_n(x)\right| \leq f(x)$$

for all the index number $n \in \mathbb{N}^+$ and $x \in S$, then f is integrable and

$$\lim_{n \to \infty} \int_S \left|f_n - f\right| d\mu = 0$$

Radamacher's theorem
If U is an open subset in $\mathbb{R}^n$, $f : U \mapsto \mathbb{R}^m$ is Lipschitz continuous, then f is differentiable almost everywhere.

Assumption 1
Let $g : \mathcal{Z} \times \mathbb{R}^d \mapsto \mathcal{X}$ be locally Lipschitz between finite dimentional vector spaces, $g_\theta(z)$ is denoted as evaluation on corrdinates (z, θ). We call g satisfies assumption 1 for a certain probability distribution $\mathbb{P}$ over $\mathcal{Z}$ if there are local Lipschitz constants $K(\theta, z)$ such that:

$$\mathbb{E}_{z \sim \mathbb{P}}\left[K(\theta, z)\right] < \infty$$

Theorem 1
Let $\mathbb{P}_r$ be a fixed distribution over $\mathcal{X}$. Let Z be a random variable (e.g Gaussian) over another space $\mathcal{Z}$. Let $g : \mathcal{Z} \times \mathbb{R}^d \mapsto \mathcal{X}$ be a function, that will be denoted $g_\theta(\boldsymbol{z})$ with $\boldsymbol{z}$ the first coordinate and $\boldsymbol{\theta}$ the second. Let $\mathbb{P}_\theta$ denote the distribution of $g_\theta(Z)$. Then:

1. If g is continuous in $\boldsymbol{\theta}$, then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is also continuous.

2. If g is locally Lipschitz and satisfies regularity assumption 1, then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere, and differentiable almost everywhere.

Proof.

1.

The goal is to show

$$\begin{aligned} &\lim_{\theta_1 \to \theta_2} \|g_{\theta_1}(\boldsymbol{z}) - g_{\theta_2}(\boldsymbol{z})\| = 0 \\ \implies &\lim_{\theta_1 \to \theta_2} \|W(\mathbb{P}_r, \mathbb{P}_{\theta_1}) - W(\mathbb{P}_r, \mathbb{P}_{\theta_2})\| = 0 \end{aligned}$$

Assume θ_1 and θ_2 are two parameter vectors in $\mathbb{R}^d$. By the definition of the Wasserstein distance,

$$\begin{aligned} W(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}) &\leq \mathbb{E}_{(a,b)\sim\gamma}\left[\|x - y\|\right] \\ &= \mathbb{E}_{(a,b)\sim\gamma}\left[\|g_{\theta_1(z)} - g_{\theta_2(z)}\|\right] \\ &= \int_{\mathcal{X}\times\mathcal{X}} \left[\|g_{\theta_1(z)} - g_{\theta_2(z)}\|\right] d\gamma \end{aligned} \tag{1}$$

If g is continuous, by the property of continuity,

$$\begin{aligned} &\lim_{\theta_1 \to \theta_2} g_{\theta_1}(z) = g_{\theta_2}(z) \\ \implies &\|g_{\theta_1}(z) - g_{\theta_2}(z)\| = 0. \end{aligned}$$

Since we assumed that $\mathcal{X}$ is compact, which means closed and bounded, there exists a constant M such that $\|g_{\theta_1}(z) - g_{\theta_2}(z)\| \leq M$ uniformly for all $\theta \in \mathbb{R}^d$ and z. By the Lebesgue's dominated convergence theorem and (1),

$$\lim_{\theta_1 \to \theta_2} \int_{\mathcal{X}\times\mathcal{X}} \left[\|g_{\theta_1(z)} - g_{\theta_2(z)}\|\right] d\gamma = \lim_{\theta_1 \to \theta_2} \mathbb{E}_z\left[\|g_{\theta_1}(z) - g_{\theta_2}\|\right] = 0$$

by the triangle inequality:

$$\left\| W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big) - W\big(\mathbb{P}_r, \mathbb{P}_{\theta_2}\big) \right\| \leq W\big(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}\big)$$

Therefore if $\theta_1 \to \theta_2$,

$$\lim_{\theta_1 \to \theta_2} W\big(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}\big) = 0$$

The continuity of $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is proved.
For proving 2, assumed g is Lipschitz locally, that is, for a given point (z, θ) in the domain of g, exists an open set $(z, \theta) \in U$ and a constant $K(z, \theta)$ such that $\forall (z', \theta') \in U$ satisfied the property,

$$\big\|g_\theta(z) - g_{\theta'}(z')\big\| \leq K(z,\theta)\big\|(\theta, z) - (\theta', z')\big\| \leq K(z,\theta)\big(\|\theta - \theta'\| + \|z - z'\|\big).$$

After taking expectation for both sides of the above equation and set $z = z'$,

$$\mathbb{E}_{z\sim\mathbb{P}_\theta}\big[\|g_\theta(z) - g_{\theta'}(z)\|\big] \leq \mathbb{E}_{z\sim\mathbb{P}_\theta}\big[K(z,\theta)\big]\big\|\theta - \theta'\big\| \qquad \forall\big(z, \theta'\big) \in U$$

define a new set

$$U_\theta := \big\{\theta' \big| (\theta', z) \in U\big\}.$$

Since U is open set, thus U_θ as well. Based on the above proof, by assumption 1 and define $K(\theta) = \mathbb{E}_{z\sim\mathbb{P}_\theta}\big[K(z,\theta)\big]$ we have:

$$\begin{aligned}\Big\|W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big) - W\big(\mathbb{P}_r, \mathbb{P}_{\theta_2}\big)\Big\| \leq & W\big(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}\big) \\ \leq & \mathbb{E}_{z\sim\mathbb{P}_\theta}\big[\|g_\theta(z) - g_{\theta'}(z')\|\big] \\ \leq & K(\theta)\big\|\theta - \theta'\big\| \qquad\qquad \forall \theta' \in U_\theta\end{aligned}$$

this implies that $W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big)$ is locally Lipschitz, and also continuous everywhere. By Radamacher's theorem, $W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big)$ must be differentiable almost everywhere. □

There is a corollary that guarantees that the Wasserstain Distance can be used as a loss function in neural networks.

Corollary 1
If g_θ is a feedforward neural network with $\boldsymbol{\theta}$ as it's parameters, that is, g_θ is a function composed by affine transformations and non-linear activation functions which are smooth Lipschitz continuous, and $\mathbb{E}_{z\sim p(z)}\big[\|z\|\big] < \infty$. Then the assumption 1 is satisfied and therefore $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere and differentiable almost everywhere.

3.3 Wasserstein GAN

The solution of Wasserstein distance' infimum form in (1) is hard to find. However, solving (1) is equivalent to solving the Kantorovich - Rubinstein duality (Villani. 2008)

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \le 1} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r(\boldsymbol{x})} - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_g(\boldsymbol{x})} \tag{5}$$

where all functions of $f : \mathcal{X} \mapsto \mathbb{R}$ are 1-Lipschitz. If we substitute $\|f\|_L \le 1$ to $\|f\|_L \le K$, then the left side of the above equations would be $K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$, and we are still in the same optimization problem. Thus, we could solve the Wasserstein distance by maximizing

$$\max_{w \in \mathcal{W}} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r(\boldsymbol{x})} \left[f_w(\boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r(\boldsymbol{x})} \left[f_w(g(\boldsymbol{z})) \right]$$

Where w are parameters for a family of functions $\{f_w\}_{w \in \mathcal{W}}$, and w belong to a compact space $\mathcal{W}$, this implies that all f_w are K- Lipschitz functions depended only on $\mathcal{W}$. In order to implement that, Arjovsky et al. proposed to regulate all weights w of the discriminator's neural network, e.g., $w \in [-0.01, 0.01]$ after updating all the weights. However, Arjovsky et al. also pointed out that weight clipping is a bad way to implement the Lipschitz constraint. Since if the clipping bound is too large, it would make a lot of extra iterations for all weights w to converge to their limit. On the contrary, if the clipping bound is too small, it would cause vanishing gradients. But still, since the Wasserstein distance is differentiable almost everywhere, ideally, the critic could be trained until optimization.

In addition, the backpropagation of G is described in the theorem below

Theorem 3 [Theorem 3 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ is a distribution, and $\mathbb{P}_g$ is another distribution over $g_\theta(Z)$ with Z as random variable, and $g_\theta(Z)$ satisfies assumption 1, then there exists a solution $f : \mathcal{X} \mapsto \mathbb{R}$ for (5). Moreover,

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{\boldsymbol{z} \sim p_g(z)} \left[\nabla_\theta f(g_\theta(\boldsymbol{z})) \right].$$

With the theory that is explained above, we have that the final WGAN algorithm they developed is like below

Algorithm 2: WGAN algorithm with default hyperparameters proposed by Arjovsky et al. Both D and G are using RMSprop as optimizer, with learning rate $\lambda = 0.00005$, weight clip c = 0.01, $n = 5$, batch size $k = 64$

```
1: while before $\theta$ converged do
2:     for training discriminator $n$ iterations do
3:         Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
4:         Sample $k$ minibatch input examples from data distribution $p_r(\boldsymbol{x})$
5:         Backpropagating all weights in $D$ by maximizing and updating the loss:
               $$g_{w_D} \leftarrow \nabla_{w_D} \frac{1}{k} \sum_{i=1}^{k} \Big[ f_w(\boldsymbol{x}^{(i)}) - f_w\big(g(\boldsymbol{z}^{(i)})\big) \Big].$$
6:         $w_D \leftarrow w_D + \lambda \cdot$ RMSProp$(w_D, g_{w_D})$
7:         $w_D \leftarrow$ clip$(w_D, [-c, c])\ \forall w_D \notin [-c, c]$
8:     end for
9:     Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
10:    Backpropagating all weights in $G$ by maximizing and updating the loss:
               $$g_{W_G} \leftarrow -\nabla_{W_G} \frac{1}{k} \sum_{i=1}^{k} \Big[ f_w\big(g(\boldsymbol{z}^{(i)})\big) \Big]$$
11:    $w_G \leftarrow w_G - \lambda \cdot$ RMSProp$(w_G, g_{w_G})$
12: end while
```

The loss function is an approximation of the Wasserstain distance.

Chapter 4

Several improved GANs

4.1 Wasserstein GAN with Gradient Penalty (WGAN-GP)

In the previous chapter, we saw that in Wasserstein GAN, Arjovsky et al introduced the new loss functions for D and G respectively

- $L(D) = -\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\big[D(\boldsymbol{x})\big] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{z})}\big[D(G(\boldsymbol{z}))\big]$
- $L(G) = \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{z})}\big[D(G(\boldsymbol{z}))\big]$

By the Kantorovich-Rubinstein duality, the weight clipping enforces a K-Lipschitz constraint. In other words, the L2 norm of D's gradient must be bounded by a constant K

$$\big\|\nabla_{\boldsymbol{x}} D(\boldsymbol{x})\big\| \leq K$$

In the previous chapter, Arjovsky et al introduced weight clipping

$$w \leftarrow \text{clip}(w, [-c, c])$$

to implement Lipschitz constrain, where c is the hyperparameter. However, as Gulrajani et al pointed out in their paper, this method has two problems:

1. The gradient can easily vanish or explode
2. Most of the weights are distributed on the clipping boundary, i.e., either equals to $-c$ or c.

In the first problem, since discriminators usually are deep neural networks, by the chain rule, if the clipping of the constant c is being set relatively small the backpropagation can cause the gradient to vanish; if c were set relatively big, the gradient can easily explode. Instead of applying weight clipping, Gulrajani et al showed that on the Swiss Roll dataset (Figure 10) the gradient penalty term can make the gradient stable through each layer.

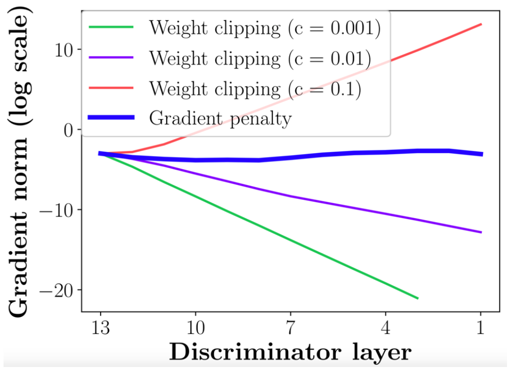


Figure 10: x axis indicate layer's index . Figure 1(b) in Gulrajani et al [18]

In the second problem, the weights are bounded on a interval $[-c, c]$, i.e., allthose weights greater than c will be set to c, all those less than $-c$ will set back to $-c$. This would cause the discriminator to learn a simple function. Gulrajani et al verified the problem on the Swiss Roll dataset and found that most of the weights take either the value c or $-c$. Instead, the gradient penalty term does not suffer from this issue, Figure 11 demonstrates this.

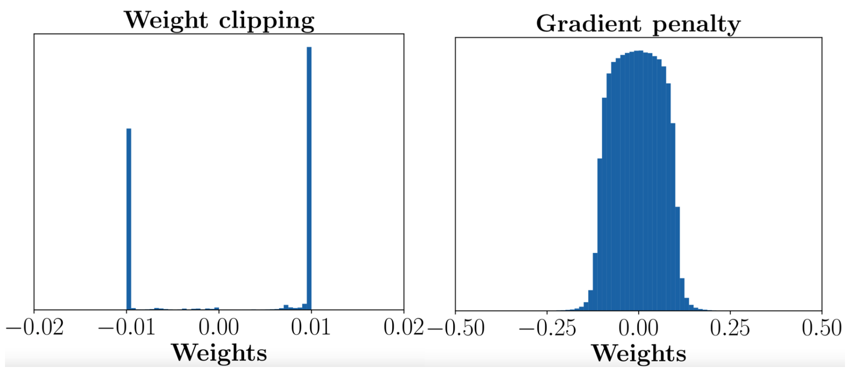


Figure 11: Figure 1(b) in Gulrajani et al [18]

The above two problems indicate that for the weight clipping method, model performance is too sensitive with respect to the corresponding hypermarameter c.

A differentiable function is K-Lipschitz if and only if the Euclidean norm of it's gradient is almost K everywhere. Instead of using weight clip, Gulrajani et al proposed to add a gradient penalty term which, as mentioned above, enforces 1-Lipschitz constraint.

$$\left(\left\|\nabla_{\boldsymbol{x}} D(\boldsymbol{x})\right\| - 1\right)^2$$

with the discriminator sufficiently trained, the norm of it's gradient will converge to 1. Then the new loss function to be minimized becomes

$$L = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g}\big[D(g(\boldsymbol{z})\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big]}_{\text{original WGAN's loss}} + \underbrace{\lambda\cdot\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\Big[\big(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1\big)^2\Big]}_{\text{new gradient penalty term}}.$$

Where $\hat{\boldsymbol{x}} = \epsilon\boldsymbol{x} + (1-\epsilon)g(\boldsymbol{z})$ and ϵ is a random real number from [0,1]. Thus any model can be penalized by the gradient penalty term if the norm of the gradient diverges from 1.

For experiment setting, Gulrajani et al proposed to not use batch normalization, but layer normalization is recommended. Below is the final algorithm:

Algorithm 4: [Algorithm 1 in Gulrajani et al [18]]. The default hyperparameters are $\lambda = 10, n = 5, \beta_1 = 0, \beta_2 = 0.9, \alpha = 0.0001$, where β_1, β_2 and α are Adam optimizer hyperparameters

```
1: while before $\theta$ converged do
2:     for training discriminator $n$ iterations do
3:         for $i = 1, \cdots,$ batch size $m$ do
4:             Sample $\boldsymbol{x}$ from data distribution $p_r(\boldsymbol{x})$
5:             Sample $\boldsymbol{z}$ from latent space $p_g(\boldsymbol{z})$
6:             Sample random number $\epsilon$ from $U[0,1]$
7:             $\hat{\boldsymbol{x}} \leftarrow \epsilon\boldsymbol{x} + (1-\epsilon)G_\theta(\boldsymbol{x})$
8:             $L^{(i)} \leftarrow D\big(G(\boldsymbol{z})\big) - D(\boldsymbol{z}) + \lambda\big(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1\big)^2$
9:         end for
10:        $w_D \leftarrow w_D - \eta\cdot$ Adam$\Big(\frac{1}{m}\sum_{i=1}^{m}\nabla_{w_D}L^{(i)}, \beta_1, \beta_2, \alpha\Big)$
11:    end for
12:    Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
11:    $w_G \leftarrow w_G - \eta\cdot$ Adam$\Big(-\frac{1}{m}\sum_{i=1}^{m}\nabla_{w_G}D\big(G(\boldsymbol{z}^{(i)}), \beta_1, \beta_2, \alpha\big)\Big)$
12: end while
```

4.2 Wasserstein GAN with A Consistency Term (CT-GAN)

Despite the fact that WGAN with gradient penalty term is able to overcome the two problems which exist in the original WGAN, Wei et al. (2017) indicate that the gradient penalty term only works on those interpolated samples $\hat{\boldsymbol{x}}$. At the early training stages, the distribution $\mathbb{P}_g$ differs a lot from $\mathbb{P}_r$. Moreover, the gradient penalty term usually fails to examine the continuity near the real sample $\boldsymbol{x}$. Therefore the discriminator can possibly fail to be constrained by 1-Lipschitz continuity.

Based on these issues, they attempt to improve the WGAN with gradient penalty by enforcing the Lipschitz continuity over the real data $\boldsymbol{x} \sim \mathbb{P}_r$. Specifically, they perturb each real sample $\boldsymbol{x}$ as $\boldsymbol{x}'$ and $\boldsymbol{x}''$, then apply Lipschitz constraint to bound the $D(\boldsymbol{x}')$ and $D(\boldsymbol{x}'')$.

Recall from the WGAN with gradient penalty, a discriminator $D : \mathcal{X} \mapsto \mathcal{Y}$ is Liptschitz continuous if and only if there exists a real number K such that $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$,

$$\|D(\boldsymbol{x}_1) - D(\boldsymbol{x}_2)\|_2 \le K \cdot \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2$$

therefore, the following soft consistency term (CT) can be added in order to penalize those violations from the above inequality.

$$CT\big|_{\boldsymbol{x}_1,\boldsymbol{x}_2} = \mathbb{E}_{\boldsymbol{x}_1,\boldsymbol{x}_2}\Big[\max\Big(\frac{\|D(\boldsymbol{x}_1) - D(\boldsymbol{x}_2)\|_2}{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2} - M', 0\Big)\Big]$$

However, Wei et al pointed out that $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is impractical, as it is hard to compute all the combinations of $(\boldsymbol{x}_1, \boldsymbol{x}_2)$. Therefore, to make the inequality able to be implemented, all $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ are assumed to be bounded by a constant M', and M' is a hyper-parameter. In addition, they proposed to apply dropout before the output of the discriminator, denoted as $D(\boldsymbol{x}')$, where $\boldsymbol{x}'$ is sampled from input data, and $D(\boldsymbol{x}'')$ is the second data point which is applied in the same corresponding way. $D\_(\cdot)$ denote the output of the second last layer of the discriminator. Thus the final consistency term and loss function is

$$CT\big|_{\boldsymbol{x}',\boldsymbol{x}''} = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[\max\big(0, \|D(\boldsymbol{x}_1) - D(\boldsymbol{x}_2)\|_2\big) + 0.1 \cdot \|(D\_(\boldsymbol{x}', D\_(\boldsymbol{x}'')\|_2 - M'\big],$$

$$L = \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g}\big[D(G(\boldsymbol{z}))\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] + \lambda_1 GP\big|_{\hat{\boldsymbol{x}}} + \lambda_2 CT\big|_{\boldsymbol{x}',\boldsymbol{x}''}.$$

Final algorithm for the CT-GAN

Algorithm 5: [Algorithm 1 in Wei et al [15]]. The default hyperparameters are $\lambda_1 = 10, \lambda_2 = 2, n = 5, \beta_1 = 0.5, \beta_2 = 0.9, \alpha = 0.0002, m = 64$ where β_1, β_2 and α are Adam optimizer hyperparameters.

```
1: while before $\theta$ converged do
2:     for training discriminator $n$ iterations do
3:         for $i$ = 1, $\cdots$, batch size $m$ do
4:             Sample $\boldsymbol{x}$ from data distribution $p_r(\boldsymbol{x})$
5:             Sample $\boldsymbol{z}$ from latent space $p_g(\boldsymbol{z})$
6:             Sample random number $\epsilon$ from $U[0,1]$
7:             $\hat{\boldsymbol{x}} \leftarrow \epsilon\boldsymbol{x} + (1-\epsilon)G_\theta(\boldsymbol{x})$
8:             $L^{(i)} \leftarrow D\big(G(\boldsymbol{z})\big) - D(\boldsymbol{z}) + \lambda_1 GP\big|_{\hat{\boldsymbol{x}}} + \lambda_2 CT\big|_{\boldsymbol{x}',\boldsymbol{x}''}$
9:         end for
10:        $w_D \leftarrow w_D - \eta\cdot$ Adam $\Big(\frac{1}{m}\sum_{i=1}^{m} \nabla_{w_D} L^{(i)}, \beta_1, \beta_2, \alpha\Big)$
11:    end for
12:    Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
11:    $w_G \leftarrow w_G - \eta\cdot$ Adam$\Big(-\frac{1}{m}\sum_{i=1}^{m} \nabla_{w_G} D\big(G(\boldsymbol{z}^{(i)}), \beta_1, \beta_2, \alpha\Big)$
12: end while
```

Chapter 5

Experiments

Conneau et al showed results trained by Vanilla GAN for some language pairs such as English(en) translated to Spanish(es), English to French(fr) and English to Esperanto(eo) etc. Their code, dictionaries and word embeddings are open source on GitHub (MUSE)[1].

Since the GPU computational resource is limited, and for the convenience of comparing the result and benchmark from the MUSE, we only select several relatively representative language pairs which Conneau et al. (2017) implemented, and plus few other language pairs.

We will describe the data we used and review the experimental procedure in details from Conneau et al first, then show our result.

5.1 data description

Facebook's AI Research department published pre-trained word vectors for 294 languages in GitHub. They trained them on Wikipedia as source text using the library called fastText. More specifically, the word vectors were trained using the skip-gram models (Bojanowski et al. 2016) with 300 as embedding dimension. All datasets can be downloaded from their GitHub repository [2].

We also tried to fit English embeddings which are trained from different corpus, these target embeddings are trained from English Common Crawl Corpus (ENC3) [3], which contains two million word embeddings.

5.2 Multilingual Unsupervised and Supervised Embeddings

In the unsupervised method of MUSE, the linear transformation W across source language embeddings and target embeddings can be learned by GANs without any given dictionary. The experimental setup will be reviewed below.

[1] MUSE: Multilingual Unsupervised and Supervised Embeddings
[2] FastText: Pre-trained word vectors
[3] NLPL word embeddings repository

5.2.1 Network architecture

The vanilla GAN's neural architecture Conneau et al. (2017) proposed is to define the generator as a linear transformation W which maps the source embedding matrix X to the target embedding matrix Y, therefore the generator is a single-layer neural network without activation function, the number of nodes in that layer depends on the embedding dimension of language pairs. Since the dimensionality of all monolingual embeddings is 300, hence the generator is a matrix with size 300×300, a square matrix. The discriminator is a multilayer perceptron with two hidden layers, both have 2048 nodes with Leaky-ReLu as activation function. The output of the discriminator is single node squeezed to probability by sigmoid.

5.2.2 Objective functions

Assume $\mathcal{X} = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(n)}\}$ and $\mathcal{Y} = \{\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(m)}\}$ are source and target language embeddings respectively, the label for source embedding is 1, 0 for target. Since the goal of the discriminator is to identify whether the input is coming from source or target. Therefore the discriminator's loss function is

$$f\mathcal{L}_D = -\frac{1}{n}\sum_{i=1}^{n} \log P_{\theta_D}\left(\text{source} = 1 \middle| W\boldsymbol{x}^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{n} \log P_{\theta_D}\left(\text{target} = 0 \middle| \boldsymbol{y}^{(i)}\right)$$

where θ_D denote the parameters of discriminator's network, $P_{\theta_D}\left(\text{source} = 1 \middle| W\boldsymbol{x}\right)$ means probability of input point coming from source embeddings, whereas $P_{\theta_D}\left(\text{target} = 0 \middle| \boldsymbol{y}\right)$ means target embeddings, i.e., real dataset.

The task of the generator is to deceive the discriminator as much as possible, thus, it's loss function is

$$\mathcal{L}_W = -\frac{1}{n}\sum_{i=1}^{n} \log P_{\theta_D}\left(\text{source} = 0 \middle| W\boldsymbol{x}^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{n} \log P_{\theta_D}\left(\text{target} = 1 \middle| \boldsymbol{y}^{(i)}\right)$$

5.2.3 Optimizer

Since they use minibatch for each training iteration and the batch size is 32, stochastic gradient descent is applied for minimizing $\mathcal{L}_D$ and $\mathcal{L}_W$, the default learning rate is 0.1 without momentum.

5.2.4 Orthogonality

Since the generator is a 300×300 matrix, Conneau et al. (2017) proposed to apply an orthogonal constraint to the generator after each minibatch iteration. It keeps the embeddings' quality after the source embedding is translated. Moreover, orthogonal matrices have a nice property, assuming $\mathcal{O}$ is an orthogonal matrix, i.e. $\mathcal{O}^T\mathcal{O} = \mathcal{O}\mathcal{O}^T = I$, it will preserve the ℓ_2 norm,

$$\|\mathcal{O}\boldsymbol{x}\|_2^2 = \langle \mathcal{O}\boldsymbol{x}, \mathcal{O}\boldsymbol{x}\rangle = \mathcal{O}^2\langle \boldsymbol{x}, \boldsymbol{x}\rangle = \|\boldsymbol{x}\|_2^2$$

This implies $\|\mathcal{O}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$, in addition this means $\mathcal{O}$ is an isometry in the $\ell 2$ space because it preserves distance.

The algorithm they used to update the transformation W is

$$W \longleftarrow (1+\beta)W - \beta(WW^T)W$$

where β is a hyperparameter, as they suggest, $\beta = 0.01$ performs well.

5.2.5 Cross-domain similarity local scaling (CSLS)

A reliable metric is crucial for the criterion of model selection, as we need to compare the similarity between source embeddings and target embeddings.

Nearest neighbors have the asymmetric property, which means that if x is the k-nn of a point y, it does not imply that y is a k-nn of x. By Radovanović et al. (2010), the asymmetric property would cause harmful results in high dimensional spaces: some points, which are called hubs, are the nearest neighbors of numerous other points, however, for some other points, which are called anti-hubs, there are no points having them as the nearest neighbors.

Therefore, Conneau et al. (2017) proposed a bi-partite neighborhood graph called Cross-domain similarity local scaling(CSLS) as the new metric. By their definition, the mean cosine similarity from a translated source embedding Wx_s to its k-target neighborhood in target space is defined as

$$r_T(Wx_s) = \frac{1}{k} \sum_{y_t \in \mathcal{N}_T(Wx_s)} \cos(Wx_s, y_t)$$

where $\mathcal{N}_T(Wx_s)$ denotes the neighborhood associated with a translated source word Wx_s, and all k words of $\mathcal{N}_T(Wx_s)$ are from target embeddings. Similarly, $\mathcal{N}_S(y_t)$ denote the neighborhood in target embedding corresponded to a target word y_t. The mean cosine similarity of a target word to its k-target neighborhood in translated source embeddings space is defined as

$$r_S(y_t) = \frac{1}{k} \sum_{Wx_s \in \mathcal{N}_S(y_t)} \cos(y_t, Wx_s)$$

these two mean cosine scores for translated source embeddings and target embeddings are implemented by the library called Faiss[4], which greatly speed up the computation of nearest neighbors. The Faiss is developed by Johnson et al. (2017) who are members of Facebook AI research group. The CSLS algorithm define as

$$\text{CSLS}(Wx_s, y_t) = 2\cos(Wx_s, y_t) - r_T(Wx_s) - r_S(y_t).$$

Obviously, there is no hyperparameter tuning requirement for it. By the interpretation of Conneau et al. (2017) , the update of CSLS algorithm increases the similarity for those anti-hub word embeddings. In contrast, it also reduces those embeddings in dense hub regions. In addition, the CSLS greatly improves the translation accuracy in experiments.

[4]Faiss: a package for efficient similarity computation.

5.2.6 Translation process

The MUSE will translate source embeddings to the target embedding space, then for each translated source embedding, select the k-nearest neighbors from target embeddings, by CSLS algorithm.

5.2.7 Evaluation procedure

Conneau et al provides some bilingual evaluation dictionaries which are translated by an internal translation tool, each dictionary translate 1500 unique source words, most of them are commonly used, also polysemy for some vocabularies, i.e., the English word "discover" translated to the Spanish words : "descubrir", "descubra", "descubre" and "descubren". After each epoch, the MUSE will calculate the CSLS accuracy on bilingual evaluation dictionary, and report the precision at 1, 5 and 10, i.e., the accuracy of nearest neighbors.

MUSE will also compute the similarity for the top 10,000 frequent words. After having finished the calculation of CSLS precision, the length of both source and target embeddings will be normalized to 1, then a subset of source word index which corresponds to the temporary dictionary will be selected (source and target word embeddings would be indexed as temporary dictionary before adversarial training), and transformed by the learned generator, to match the target word by k-nearest neighbor and CSLS algorithm. At the end of the epoch, the best model will be saved based on the mean cosine CSLS for the maximum of 10,000 top frequent embeddings.

5.2.8 Procruster refinement

Use those most frequent words as anchor points, then apply Procruster algorithm, i.e., compute the SVD(YX^T), where X and Y is source and target embeddings respectively, then use U and V to solve

$$W^* = \underset{W}{\text{argmin}} \, \|WX - Y\|_F = UV^T,$$

5.3 Experimental set up

In our experiment, we focused on three improved GANs, i.e., Wasserstain GAN, WGAN with gradient penalty and CT-GAN. Our code is based on MUSE, in which we are using Pytorch as deep learning library instead of TensorFlow. We later found out that adding layer normalization to the output of the fully connected layer not only increases the accuracy, but also stabilizes the training process. Therefore, we will later discuss all the three lineages of WGANs with layer normalization, and without layer normalization. Moreover, we are only focusing on the results obtained from adversarial training instead of after Procruster refinement.

5.3.1 Target language selection

We fixed English as the source language, and selected all the other languages from the table 2 of Søgaard et al (2018) as target languages, which are Spanish(ES), Estonian(ET), Greek(EL), Finnish(FI), Hungary(HU) and Polish(PL), plus Chinese(ZH), which belong to a different language family, and English whose embeddings are trained from a different corpus.

5.3.2 Hyper-parameter search

Since the hyper-parameter search costs lot of GPU resources, we decide to only tune those main hyper-parameters which mostly affect our models. For those secondary hyper-parameters such as β_1, β_2 and α in Adam optimizer, we are using the default values from PyTorch.

We first used grid search to find relatively good hyper-parameters from English to Spanish and Estonian for all three different Algorithms, then extend to other languages pairs.

For Wasserstain GAN, we found $c = 0.25$, and RMSprop with learning rate 0.0005 have best performance .

For WGAN with gradient penalty, we use $\lambda = 0.5$, and Adam with learning rate 0.0005.

For CT-GAN, we use $\lambda_{GP} = 0.5, \lambda_{CT} = 2$ and $m = 0.2$.

5.3.3 Define the metric

We use two types of metrics which we mentioned at the previous section to select models and hyper-parameters.

- The CSLS nearest neighbor precision at 1, 5 and 10, which evaluate the performance on the evaluation dictionary.
- The mean cosine CSLS, which measure the similarity of the top frequent 10,000 words.

In table 5.1, the source language is English, we can see that the mean cosine CSLS inaccurately reflect translation performance, e.g. for English to Chinese(zh) and Estonian(et), the precision 10 is very low, but has high overall mean CSLS, this indicates that the translated embeddings have very close neighbors, but almost all of them are incorrect translations. Therefore, we use CSLS nearest neighbor (P@1,5 and 10) to evaluate translation performance.

Vanilla	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
p@1	0.0	39.8	3.2	13.9	10.3	19.1	16.4	20.3	0.0
p@5	0.0	67.8	9.8	28.4	26.3	38.6	31.8	46.5	0.0
p@10	0.0	75.1	14.2	36.4	35.8	48.1	39.9	57.4	0.0
mean csls	47.9	67.1	49.6	56	56.1	56.8	55.5	58.6	79.6

Table 5.1: p@5, p@10 and mean cosine CSLS for differen target languages

5.3.4 Network architecture

We keep using the same network architecture for all three improved GANs, i.e., two hidden layers with 2048 dimensionality, but, different loss functions from Vanilla GAN, and without sigmoid to squeeze discriminator's output to probability. Plus added layer normalization to each layers.

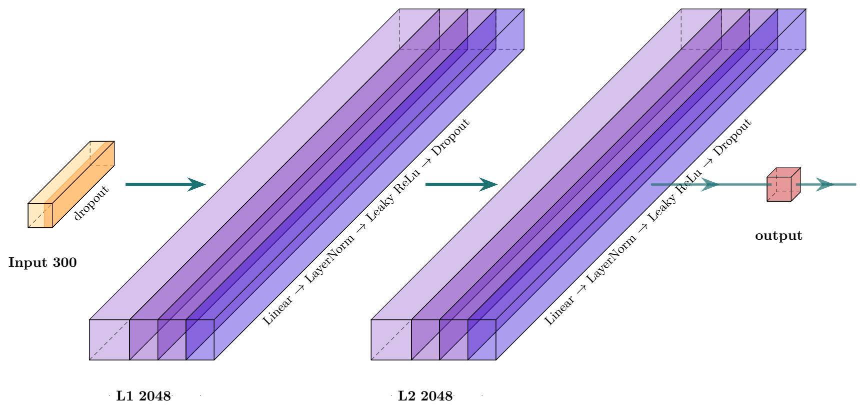

dropout
Input 300
Linear → LayerNorm → Leaky ReLu → Dropout
L1 2048
Linear → LayerNorm → Leaky ReLu → Dropout
L2 2048
output

Figure 5.1: Architecture of discriminator

5.3.5 Stability test

Since the performance depends on how the weights are initialized, e.g., an experiment can produce a good result, but in the same setting with different random weight initialization, the generator may fail. Therefore, for some algorithms, we fixed hyper-parameters and let ten different random seeds vary, to test how stable these GANs are. In addition, we defined a run as failed if the corresponding P@10 is less than 10%.

5.3.6 Benchmark

We tried to reproduce the result in table 1 of Conneau et al , but we could not get the same result. This may be due to the best random seed not being specified, hence we implement the same setting from their paper, and use the new reproduced results as our benchmark.

5.4 Results

Since we later found that added layer normalization in each of discriminator's layer can moderately increase the performance, therefore we first report the results of three improved GANs with layer normalization, and later discuss each improved GAN individually. We also found that instead of exponentially shrinking the learning rate for each epochs, fix the learning rate until finished the training process performs better.

WGAN	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
p@1	0.0	40.4	1	7.7	4.1	16.5	0.0	0.0	0.0
p@5	0.1	68.6	3.5	19.2	12.4	34.8	0.0	0.0	0.1
p@10	0.1	75.6	6.2	26.5	17	43.5	0.1	0.1	0.1
WGAN-GP	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
p@1	83.7	36.3	11.2	13.2	14.5	23	16.2	19.6	5.7
p@5	89.3	64.3	24.2	29.5	34.3	44	31.1	43.8	12.6
p@10	90.8	71.3	30.9	36.6	42.3	52.5	38.5	53.8	16
WGAN-GP	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
p@1	83.7	36.3	11.2	13.2	14.5	23	16.2	19.6	5.7
p@5	89.3	64.3	24.2	29.5	34.3	44	31.1	43.8	12.6
p@10	90.8	71.3	30.9	36.6	42.3	52.5	38.5	53.8	16

Table 5.2: p@1 and p@5, p@10 , random seed is -1

We can see that WGAN with gradient penalty (table 5.3) works best and quilt stable, and the accuracy of English translate to itself is very high, even to translate to Chinese, WGAN-GP still learned part of it's distribution instead of completely failed as other three GANs. In the results of vanilla GAN (table 5.1), the training from en to en and zh are completely failed, even for en to et which is not a far away language, it still barely learned small distribution from it.

the original WGAN (table 5.3) works not as good as we expect.

5.4.1 stability test

We changed ten different random seeds for all the four algorithms, in order to test how weight initialization influence the training process.

Vanilla	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
Failed	10	1	9	0	0	1	1	0	10
max p@1	0.1	39.6	5.7	13.7	13.5	25.6	17.2	24.9	0.0
min p@1	0.0	37.9	0.0	12.0	9.1	21.5	10.9	17.3	0.0
avg p@1	0.0	39.2	0.6	12.2	11.1	22.7	15.0	19.5	0.0
std p@1	0.0	0.7	1.7	1.0	1.6	1.2	2.1	2.6	0.0
max p@5	0.1	67.7	15.6	30.0	31.7	45.2	34.0	50.7	0.1
min p@5	0.1	63.8	0	25.1	23.4	39.6	24.5	40.1	0.0
avg p@5	0.1	66.8	1.6	27.4	27.5	42.4	30.1	43.4	0.0
std p@5	0.0	1.2	4.7	1.8	2.5	1.6	3.1	3.4	0.0
max p@10	0.4	74.9	22.4	37.7	65.7	54.5	0.103	61.0	0.1
min p@10	0.1	71.4	0	31.8	31.3	47.5	30.9	50.3	0.0
avg p@10	0.1	74.1	2.3	34.8	35.5	51.3	37.5	53.8	0.1
std p@10	0.1	1.1	6.7	2.0	2.9	1.8	3.7	3.5	0.0

Table 4: result of ten runs Vanilla GAN with differen random seed

WGAN	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
Failed	10	2	9	10	10	4	5	4	8
max p@1	0.0	38.7	6.0	0.5	1.5	20.4	8.9	17.7	6.1
min p@1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
avg p@1	0.0	30.4	0.9	0.1	0.2	9.1	3.5	8.8	1.2
std p@1	0.0	15.2	1.9	0.2	0.4	7.9	3.5	7.4	2.4
max p@5	0.1	65.4	14.7	1.2	5.1	22.0	20.9	39.1	13.6
min p@5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
avg p@5	0.0	52.1	2.2	0.3	0.8	16.5	9.0	20.5	2.7
std p@5	0.0	26.0	4.6	0.3	1.5	8.3	8.8	17.0	5.3
max p@10	0.1	73.0	21.0	2.2	8.0	29.1	28.7	49.1	18.5
min p@10	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.1
avg p@10	0.0	58.0	3.2	0.5	1.3	22.1	12.3	26.3	3.7
std p@10	0.0	28.9	6.6	0.6	2.3	11.1	11.9	21.6	7.1

Table 4: result of ten runs WGAN original for different random seed

WGAN-GP	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
Failed	1	0	0	0	0	1	0	0	0
max p@1	86.9	35.5	11.0	14.2	16.0	23.7	15.4	18.6	6.9
min p@1	0.0	34.6	10.3	6.7	13.4	0.0	11.9	16.2	3.4
avg p@1	76.3	35.0	9.9	8.6	14.5	19.7	14.7	17.8	5.3
std p@1	25.6	0.4	1.1	2.0	1.0	6.7	1.4	0.7	1.2
max p@5	91.8	62.2	24.0	31.0	36.0	44.9	32.6	43.8	14.2
min p@5	0.1	59.7	21.0	17.9	31.5	0	27.4	40.9	9.1
avg p@5	80.9	61.3	22.5	20.7	33.6	38.1	30.6	42.6	11.9
std p@5	27.0	0.8	1.2	3.6	1.4	12.8	1.7	0.9	1.9
max p@10	93.6	69.7	31.7	38.3	44.5	53.5	40.9	54.5	18.6
min p@10	0.4	67.5	27.0	23.3	39.5	0.1	35.6	50.9	12.0
avg p@10	82.1	68.7	29.1	26.9	42.1	45.8	38.3	52.7	15.3
std p@10	27.3	0.6	1.3	4.0	1.5	15.3	1.5	1.2	2.1

Table 4:WGAN-GP with ten runs different random seed

CT-GAN	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
Failed	7	0	0	0	1	0	0	0	10
max p@1	71.9	32.1	5.8	10.5	10.9	15.7	10.8	13.7	0.1
min p@1	0.0	28.8	4.2	8.3	0.0	13.4	8.9	12.5	0.0
avg p@1	21.9	30.3	5.2	9.2	9.1	15.3	9.4	13.3	0.0
std p@1	32.8	1.2	0.7	1.1	3.1	0.9	0.9	0.5	0.0
max p@5	80.7	54.8	15.7	23.9	26.9	34.8	23.8	34.2	0.4
min p@5	0.0	51.3	12.5	19.2	0.1	30.4	20.2	30.5	0.0
avg p@5	24.7	53.3	13.9	21.8	22.2	33.2	22.0	32.6	0.2
std p@5	36.2	1.3	1.1	1.8	7.5	1.4	1.2	1.1	0.1
max p@10	83.2	63.0	22.0	30.4	35.6	44.6	30.5	44.0	0.7
min p@10	0.0	58.7	17.4	24.4	0.1	39.1	25.7	39.9	0.1
avg p@10	25.6	61.0	19.6	27.8	28.9	41.4	28.4	41.8	0.3
std p@10	36.8	1.3	1.4	2.0	9.7	1.6	1.5	1.1	0.1

Table 4: result of ten runs WGAN original for different random seed

5.4.2 WGAN-Original

WGAN	**en**	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
p@1	0.0	20.4	0.1	0.0	0.0	0.0	0.0	10.5	0.0
p@5	0.1	38.6	0.1	0.0	0.0	0.0	0.0	26.9	0.0
p@10	0.3	44.7	0.2	0.1	0.1	0.0	0.1	35.1	0.0

Table 4: p@1 and p@5, p@10 from different target languages

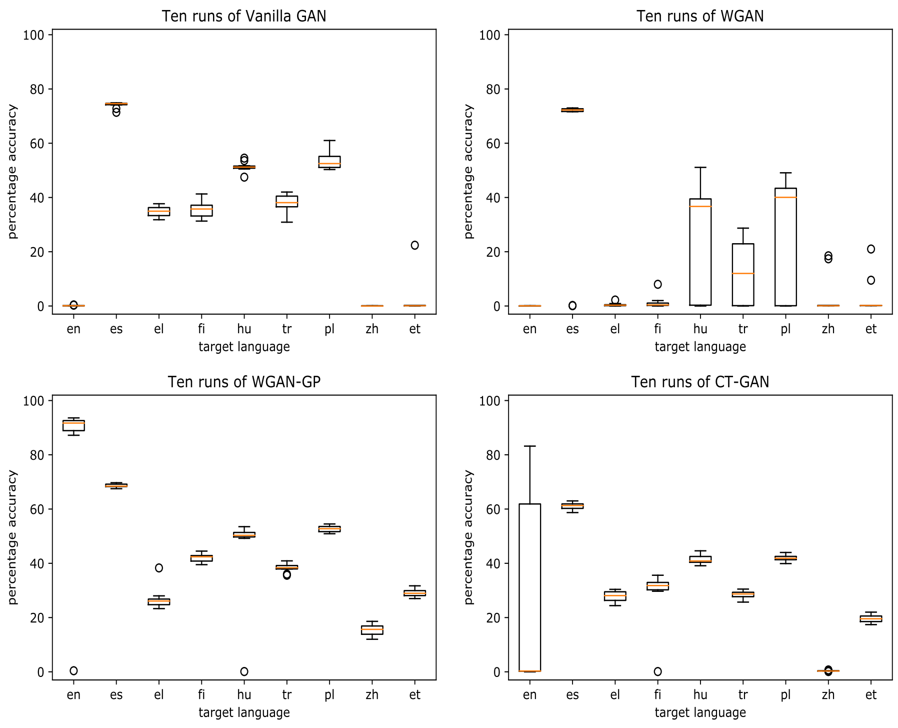


Figure 5.5: 2d T-SNE after extract 150 components by PCA

5.4.3 Nearest neighbor table

tgt	Vanilla GAN	WGAN
EN	place place place place place	place place place place place
	0.0	0.0
	0.0	14.0
	0.1	9.2
	0.1	9.2
ES	0.9	39.2
	0.1	0.1
	0.2	26.7
	0.2	17.4
	0.2	17.4
ET	1.2	47.2
	0.1	0.1
	0.4	31.7
	0.3	20.5
	0.3	20.5

Table 4: result of ten runs WGAN original for different random seed

tgt	**WGAN-GP**
EN	cat, cats, dog, rabbit, kitten, pet, kitty
	denmark, norway, sweden, netherlands, copenhagen, finland, danish
	relation, relationship, relate, regard, analogous, namely, relating
	smoke, fumes, cigarette, smoking, fire, burning, soot
	specific, relevant, particular, appropriate, specifically, related, specify
ES	gato, perro, gorila, cachorro, gatos, ratón, conejo
	dinamarca, suecia, noruega, copenhague, finlandia, danesa, escania
	relaciona, relacionar, relación, concierne, relacionan, relacionada, respecto
	humo, humos, fumar, cigarrillo, arder, fuego, inflamable
	específica, específicas, específicos, específico, determinada, ... concreta, determinadas
ET	1.2
	0.1
	0.4
	0.3
	0.3

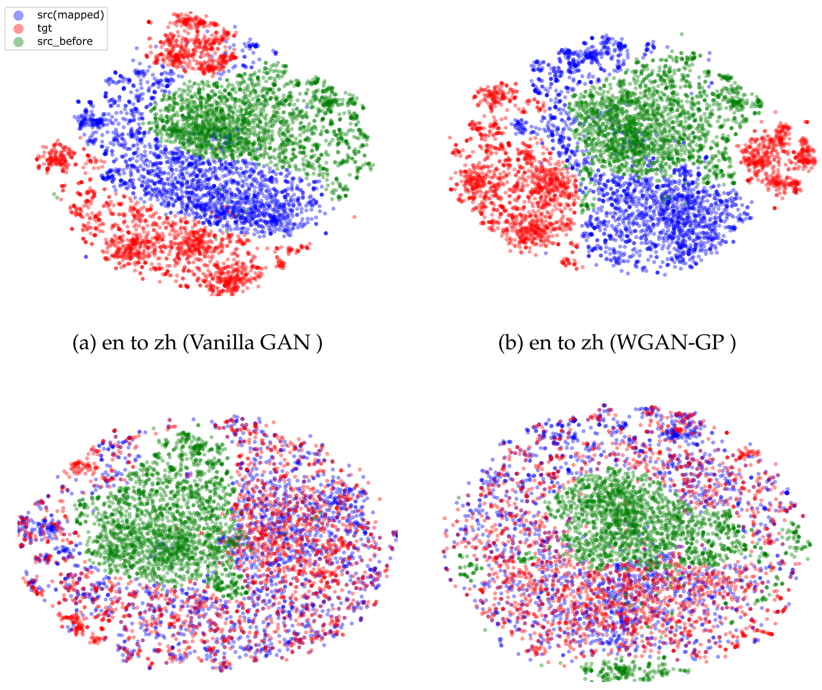
Table 4: translations of "cat", "degree", "denmark", "smoke" and "relation", K = 7

5.4.4 Overview visualization

T-Distributed Stochastic Neighbor Embedding (T-SNE) is a dimensionality reduction technique, which is a non-linear manifold learning algorithm. It developed by (Maaten et al. 2008). Compare with PCA, T-SNE more accurately captures main information from high dimensional space than PCA does. However, T-SNE costs much more computational time than PCA, hence, for visualize the performance of GANs worked on some languages pairs, we first extract the most 150 relevant components from 300 by PCA, then transform those best 150 components to 2d space by T-SNE.

We randomly sample 2000 points out of the most frequent 10,000 word embeddings from

- Original source embeddings before transformed (green dots).
- Source embeddings transformed by generator (blue dots).
- Target embeddings (red dots).



(a) en to zh (Vanilla GAN)

(b) en to zh (WGAN-GP)

(c) en to es (WGAN-GP)

(d) en to en (WGAN-GP)

Figure 5.5: 2d T-SNE after extract 150 components by PCA

For English translate to relatively closed embeddings such as Spanish and English which is trained from different Corpus, we can see that from figures 5.5(a) and (b), the translated English embeddings are mixed very well with target embeddings, but for English translate to largely different language such as Chinese as showed in the figures 5.5(a) and (b), the discriminator successfully separate almost all samples, at least in the circumstance of linear generator.

5.4.5 drawback of the three new WGANs

maybe there is no linear transformation between those languages, because EN to EN ,future work maybe remove the assumption, change the generator to non-linearity

5.4.6 Final experiment

- translate some most frequent words,
- translate some rare words.
- compare how model performs on frequent words, and rare words.

Chapter 6

Discussion

We can see that WGAN with gradient penalty (table 5.3) works best and quilt stable, and the accuracy of English translate to itself is very high, even to translate to Chinese, WGAN-GP still learned part of it's distribution instead of completely failed as other three GANs. In the results of vanilla GAN (table 5.1), the training from en to en and zh are completely failed, even for en to et which is not a far away language, it still barely learned small distribution from it.

the original WGAN (table 5.3) works not as good as we expect.

we can see that WGAN-GP is not easily suffered from different dataset, since for en to en and zh, vanilla GAN completely failed

6.1 Future work

investigate the non-linear transformation,

Appendix A

Some deep learning techniques

A.1 Layer normalization

Training a deep neural network can cause internal covariate shift, i.e., the change of the input distribution of all layers. Especially for a deep neural network, even a small change can be expanded when the network goes deeper, therefore vanishing gradients can easily happen for saturating non-linearities. Batch normalization can help to reduce internal covariate shift by normalizing mini-batch input across their corresponding size. In particular,

$$\begin{aligned}
\mu_j &= \frac{1}{m}\sum_{j=1}^{n} \boldsymbol{x}_{ij} \\
\sigma_j^2 &= \frac{1}{m}\sum_{j=1}^{m} (\boldsymbol{x}_{ij} - \mu_i)^2 \\
\hat{\boldsymbol{x}} &= \frac{\boldsymbol{x}_{ij} - \mu_i}{\sqrt{\sigma^2 + \epsilon}},
\end{aligned}$$

where i indicates the index of example, and j is the index of feature. Thus the network converges relatively faster. The batch normalization method is proposed by Loffe et al. (2015).

Despite it achieving great success, batch normalization still has two main drawbacks:

- Model performance depends on the mini-batch size, because the mean and the variance ,which are calculated by mini-batch, cause errors and these errors will vary between different mini-batches.
- It's very difficult to apply to RNN.

Layer normalization is proposed by Ba et al. (2016). Unlike batch normalization, layer normalization does not depend on the size of the mini-batch, and also works well on RNNs. For each input example, it normalizes across features in a layer, more specifically,

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} \boldsymbol{x}_{ij}$$
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m} (\boldsymbol{x}_{ij} - \mu_i)^2$$
$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_{ij} - \mu_i}{\sqrt{\sigma^2 + \epsilon}}$$

A.2 RMS prop

RMSprop is a optimization algorithm of gradient descent. It is proposed in the lecture 6 of Hinton et al [5] online course, and it is unpublisheded.

The motivation for developing RMSprop is to solve Adagrad's vanishing learning date problem. Unlike other gradient descent algorithm such as sgd or mini-batch gradient descent, by Ruder's explanation, Adagrad can adapts learning rate during training process, but the denominator term tends to zero after some training iterations. In particular, the RMSprop is

$$\mathbb{E}\left[\boldsymbol{g}^2\right]_t = 0.9\mathbb{E}\left[\boldsymbol{g}^2\right]_{t-1} + 0.1\boldsymbol{g}_t^2$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\mathbb{E}\left[\boldsymbol{g}^2\right]_t + \epsilon}}\boldsymbol{g}_t$$

where the ϵ is a small positive number to prevent division equals to zero. The learning rate divided by the mean of gradients which is exponentially decreasing.

A.3 Adam

Adam (Adaptive Moment Estimation) is another popular adaptive gradient descent algorithm, it proposed by Kingma et al. (2015). it combined both RMSprop and momentum, i.e. it not only compute and save the exponentially decaying of previous mean squared gradient, but also compute the previous momentum. More specifically,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\boldsymbol{g}_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$$

[5] Neural Networks for Machine Learning

When m_t and v_t are initialized to zero vectors, Kingma et al. found that the they are unbiased. The final bias corrected terms are

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

combined both

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\hat{v}}+\epsilon}\hat{m}_t$$

Where the default settings are $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. In practice, Adam performs better than other adaptive gradient descent algorithm.

A.4 leaky Relu

The leaky Relu is an non-linear activation, which is developed by Maas et al. (2013). Since the normal Relu is $f(x) = \max(0, x)$, when $f(x)$ is negative, it's gradient will be zero. Therefore, if the output of $f(x)$ mostly are zeros, the normal Relu can cause vanishing gradients. The leaky Relu is to solve this problem by multiply a positive number which is greater than 1, more specifically,

$$f(x) = \begin{cases} x & x \geq 0 \\ \frac{x}{a} & 0 < x \end{cases}$$

where $a \in (1, +\infty)$, thus, the leaky Relu slightly change the gradient of the negative part of $f(x)$'s output, and make the backpropagation not suffered by vanishing gradients.

Bibliography

[1] Sebastian Ruder, Ivan Vulić, and Anders Søgaard. A Survey Of Cross-lingual Word Embedding Models. 2017.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013.

[3] Ian J.Goodfellow, Jean Pouget-Abadue, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. 2014.

[4] Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the Automatic Anime Characters Creation with Generative Adversarial Networks. 2017.

[5] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. 2016.

[6] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralbai. Generating Videos with Scene Dynamics. 2016.

[7] William Fedus, Ian Goodfellow, and Andrew M. Dai. MaskGAN: Better Text Generation via Filling in the______. 2018.

[8] Olof Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. 2016.

[9] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative Adversarial Text to Image Synthesis. 2016.

[10] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word Translation Without Parallel Data. 2018.

[11] Urysohn's Lemma - Wikipedia

[12] Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. Normalized Word Embedding and Orthogonal Transform for Bilingual Word Translation. 2015.

[13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. 2017.

[14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. 2017.

[15] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, Liqiang Wang. Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect. 2017.

[16] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. 2017.

[17] Cédric Villani. Optimal transport, old and new. 2008.

[18] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin and Aaron Courville. Improved Training of Wasserstein GANs. 2017.

[19] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. 2016.

[20] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. 2017.

[21] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. 2010.

[22] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. Hubs in Space: Popular Nearest Neighbors in High-Dimensional Data. 2010.

[23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. 2017.

[24] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015.

[25] Anders Søgaard, Sebastian Ruder and Ivan Vulić. On the Limitations of Unsupervised Bilingual Dictionary Induction. 2018.

[26] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton. Layer Normalization. 2016.

[27] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2017.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2015.

[29] Andrew L. Maas , Awni Y. Hannun and Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. 2013

[30] Laurens van der Maaten and Geoffrey Hinton Visualizing data using t-SNE. 2008